METHOD FOR INVOKING AND INTEGRATING MULTIPLE FUNCTIONAL MODULES

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present invention is related to the commonly owned, co-pending U.S. patent application 10/083,075, entitled "Application Portability And Extensibility Through Database Schema And Query Abstraction," filed February 26, 2002, and commonly owned co-pending application, entitled "Dynamic Functional Module Availability," filed herewith (Attorney Docket No. ROC920030277US1), which are herein incorporated by reference.

BACKGROUND OF THE INVENTION

Field of the Invention

[0002] The present invention generally relates to data processing and more particularly to provide a method for invoking and integrating multiple functional modules.

Description of the Related Art

[0003] Databases are computerized information storage and retrieval systems. A relational database management system is a computer database management system (DBMS) that uses relational techniques for storing and retrieving data. The most prevalent type of database is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways. A distributed database is one that can be dispersed or replicated among different points in a network. An object-oriented programming database is one that is congruent with the data defined in object classes and subclasses.

[0004] Regardless of the particular architecture, in a DBMS, a requesting entity (e.g., an application or the operating system) demands access to a specified database by issuing a database access request. Such requests may include, for instance, simple catalog lookup requests or transactions and combinations of transactions that operate to read, change and add specified records in the database. These requests are made using high-level query languages such as the Structured Query Language (SQL). Illustratively, SQL is used to make interactive queries for getting information from and updating a database such as International Business Machines' (IBM) DB2, Microsoft's SQL Server, and database products from Oracle, Sybase, and Computer Associates. The term "query" denominates a set of commands for retrieving data from a stored database. Queries take the form of a command language that lets programmers and programs select, insert, update, find out the location of data, and so forth.

[0005] One of the issues faced by data mining and database query applications, in general, is their close relationship with a given database schema (e.g., a relational database schema). This relationship makes it difficult to support an application as changes are made to the corresponding underlying database schema. Further, the migration of the application to alternative underlying data representations is inhibited. In today's environment, the foregoing disadvantages are largely due to the reliance applications have on SQL, which presumes that a relational model is used to represent information being queried. Furthermore, a given SQL query is dependent upon a particular relational schema since specific database tables, columns and relationships are referenced within the SQL query representation. As a result of these limitations, a number of difficulties arise.

[0006] One difficulty is that changes in the underlying relational data model require changes to the SQL foundation that the corresponding application is built upon. Therefore, an application designer must either forgo changing the underlying data model to avoid application maintenance or must change the application to reflect changes in the underlying relational model. Another difficulty is that extending an application to work with multiple relational data models requires separate versions of the application to reflect the unique SQL requirements driven by each unique relational schema. Yet another difficulty is evolution of the application to work with alternate data representations because SQL is designed for use with relational systems. Extending the application to support alternative data representations, such as XML, requires rewriting the application's data management layer to use non-SQL data access methods.

[0007] A typical approach used to address the foregoing problems is software encapsulation. Software encapsulation involves using a software interface or component to encapsulate access methods to a particular underlying data representation. An example is found in the Enterprise JavaBean (EJB) specification that is a component of the Java 2 Enterprise Edition (J2EE) suite of technologies. In accordance with the EJB specification, entity beans serve to encapsulate a given set of data, exposing a set of Application Program Interfaces (APIs) that can be used to access this information. This is a highly specialized approach requiring the software to be written (in the form of new entity EJBs) whenever a new set of data is to be accessed or when a new pattern of data access is desired. The EJB model also requires a code update, application built and deployment cycle to react to reorganization of the underlying physical data model or to support alternative data representations. EJB programming also requires specialized skills since more advanced Java programming techniques are involved. Accordingly, the EJB approach and other similar approaches are rather inflexible and costly to maintain for general-purpose query applications accessing an evolving physical data model.

[0008] Another shortcoming of the prior art, is the manner in which data is processed prior to being presented to the user. A number of software solutions support the use of multiple functional modules to process data as desired by the user, but management of functional modules execution is difficult. For example, a query building tool will present the user with a list of functional modules that aid in the analysis of query results. Often times, execution of numerous functional modules are needed to compile the data in the desired state. Unfortunately, the selected functional modules need to be invoked individually by the user. This can be a very inconvenient and inefficient process for invoking multiple functional modules.

[0009] Current workflow technology provides the ability to call multiple functional modules in a specified order, but there is an accompanying drawback: users are required to perform data transformation each time data passed from one functional module to another. For example, if four functional modules, FM1, FM2, FM3, and FM4 are called (respectively) and each successive functional depends on the result set produced by the functional module executed immediately prior to it, data transformation would need to be performed by the user three separate times: between FM1 and FM2, FM2 and FM3, FM3 and FM4.

[0010] Users typically employ two methods for performing the data transformation. One method comprises creating a custom program, or functional module, for extracting data from the result set produced by the first functional module and then formatting it in accordance with the requirements of the next functional module to be executed. For example, a custom program would be used to transform the result set produced by FM1 and prepare the data to be passed as input to FM2. Of course, this would need to happen with data produced by FM2, and again with FM3's result set. Another method consists of utilizing mapping tools to allow for the mapping of data fields from one program to the next. For example, the mapping tool would allow the user to map the output fields of FM1 to the input fields of FM2. The fields are mapped by users prior to execution of the programs. At runtime, data is transformed per the field mapping definitions. Both of these methods for performing data transformation are cumbersome and inefficient to use and depend heavily on user interaction.

[0011] Therefore, there is a need for an improved method for executing and integrating multiple functional modules without requiring custom programming or a large amount of user interaction. In addition, the method should be easy to employ, and should not add to the complexity of implementing an integrated solution.

SUMMARY OF THE INVENTION

[0012] The present invention generally provides methods, articles of manufacture and systems for automatically invoking a set of user-specified functional modules from within an application.

[0013] One embodiment provides a method for automatically invoking a plurality of functional modules from within an application. The method generally includes providing an interface allowing a user to specify the plurality of functional modules, providing a configuration file containing information regarding invocation of the functional modules, and invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file.

[0014] Another embodiment provides a method for automatically invoking a plurality of specified functional modules from within an application. The method generally includes (a) obtaining a set of one or more parameters required for invoking the specified functional modules, (b) invoking one or more of the specified functional modules whose required parameters are available in a result set collection, (c) obtaining a result set in response to invoking the one or more functional modules, (d) adding the result set to the result set collection, and (e) repeating steps (a) - (d) until all the specified functional modules have been executed.

[0015] Another embodiment provides a computer readable medium containing a computer readable medium containing a program for automatically invoking and integrating a plurality of functional modules from within an application. When executed, the program performs operations generally including providing an interface allowing a user to specify the plurality of functional modules, providing a configuration file containing information regarding invocation of the functional modules, and invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file.

[0016] Another embodiment provides a system for automatically invoking and integrating a plurality of functional modules comprising a plurality of functional modules, a configuration file containing information regarding execution of the functional modules, and an application from which the functional modules are accessible. The application is generally configured to provide an interface allowing a user to specify a set of functional modules and execute the functional modules in a manner determined according to information retrieved from the configuration file.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] So that the manner in which the above recited features, advantages and objects of the present invention are attained and can be understood in detail, a more particular description of the invention, briefly summarized above, may be had by reference to the embodiments thereof which are illustrated in the appended drawings.

[0018] It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments.

[0019] FIG. 1 is a computer system illustratively utilized in accordance with the present invention.

[0020] FIGs 2A and 2B provide a relational view of software components, including plug-in components and XML configuration files, of one embodiment of the present invention.

[0021] FIG. 3A is a flow chart illustrating exemplary operations utilizing an explicit sequence for plug-in execution, according to aspects of the present invention.

[0022] FIG. 3B is a flow chart illustrating exemplary operations utilizing a derived sequence for plug-in execution, according to aspects of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0023] The present invention generally is directed to methods, articles of manufacture and systems for invoking multiple functional modules from within an application. The functional modules may be invoked in a prescribed or derived sequence, without requiring data transformation.

[0024] As used herein, the phrase functional module generally refers to a set of coded instructions that enable a computer to perform a specified function. Plug-in components, analysis routines, functions or programs among other terms may constitute functional modules. Further, functional modules may be implemented internally or externally to a system, while remaining accessible by that system. While a functional module may exist in any or all of these forms, to facilitate understanding, the term plug-in will be used to refer to any functional module described herein. While the following description focuses on selecting the subset of plug-ins related to an application designed for the building and initiating of a query, those skilled in the art will recognize the methods described herein may be used with any applications that utilize plug-ins or other type of functional modules.

[0025] As used herein, the term metadata refers to descriptive information including the attributes of functional modules and result set data objects. Metadata associated with functional modules includes the number and type of required input and output (I/O) parameters and security requirements. Metadata may also comprise detailed information describing result sets returned from functional modules, such as column names, datatypes of columns, number of records returned, and content.

[0026] Further, as used herein, the term user may generally apply to any entity utilizing the data processing system described herein, such as a person (e.g., an individual) interacting with an application program or an application program itself, for example, performing automated tasks. While the following description may often refer to a graphical user interface (GUI) intended to present information to and receive information from a person, it should be understood that in many cases, the same functionality may be provided through a non-graphical user interface, such as a command line and, further, similar information may be exchanged with a non-person user via a programming interface.

EXEMPLARY APPLICATION ENVIRONMENT

[0027] FIG. 1 shows an exemplary networked computer system 100, in which embodiments of the present invention may be utilized. For example, embodiments of the present invention may be implemented as a program product for use with the system 100, to invoke and integrate a set of functional modules (hereinafter generically referred to as plug-ins) 162 as specified by the corresponding XML configuration file 160. The user (e.g., a user of an application 120 running on a client computer 102) may configure the XML configuration file 160 via the interface 122. The exact functions performed by the plug-ins may vary. For example, certain plug-ins may facilitate query building, while others provide printing support, or perform data analysis. Plug-ins 162 that perform data analysis often produce result sets packaged in result set data objects 165. These data objects 165 may comprise result data along with metadata, such as field attributes, associated with the result set.

[0028] The program(s) of the program product defines functions of the embodiments (including the methods described herein) and can be contained on a variety of signal-bearing media. Illustrative signal-bearing media include, but are not limited to: (i) information permanently stored on non-writable storage media (*e.g.*, read-only memory devices within a computer such as CD-ROM disks readable by a CD-ROM drive); (ii) alterable information stored on writable storage media (*e.g.*, floppy disks within a diskette drive or hard-disk drive); or (iii) information conveyed to a computer by a communications medium, such as through a computer or telephone network, including wireless communications. The latter embodiment specifically includes information downloaded from the Internet and other networks. Such signal-bearing media, when carrying computer-readable instructions that direct the functions of the present invention, represent embodiments of the present invention.

[0029] In general, the routines executed to implement the embodiments of the invention, may be part of an operating system or a specific application, component, program, module, object, or sequence of instructions. The software of the present invention typically is comprised of a multitude of instructions that will be translated by the native computer into a machine-readable format and hence executable instructions. Also, programs are comprised of variables and data structures that either reside locally to the program or are found in memory or on storage devices. In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. However, it should be appreciated that any particular nomenclature that follows is used merely for convenience, and thus the invention should not be limited to use solely in any specific application identified and/or implied by such nomenclature.

[0030] As illustrated in FIG. 1, the system 100 generally includes client computers 102 and at least one server computer 104, connected via a network 126. In general, the network 126 may be a local area network (LAN) and/or a wide area network (WAN). In a particular embodiment, the network 126 is the Internet.

[0031] As illustrated, the client computers 102 generally include a Central Processing Unit (CPU) 110 connected via a bus 130 to a memory 112, storage 114, an input device 116, an output device 119, and a network interface device 118. The input device 116 can be any device to give input to the client computer 102. For example, a keyboard, keypad, light-pen, touch-screen, track-ball, or speech recognition unit, audio/video player, and the like could be used. The output device 119 can be any device to give output to the user, e.g., any conventional display screen. Although shown separately from the input device 116, the output device 119 and input device 116 could be combined. For example, a client 102 may include a display screen with an integrated touch-screen or a display with an integrated keyboard.

[0032] The network interface device 118 may be any entry/exit device configured to allow network communications between the client 102 and the server 104 via the network 126. For example, the network interface device 118 may be a network adapter or other network interface card (NIC). If the client 102 is a handheld device, such as a personal digital assistant (PDA), the network interface device 118 may comprise any suitable wireless interface to provide a wireless connection to the network 126.

[0033] Storage 114 is preferably a Direct Access Storage Device (DASD). Although it is shown as a single unit, it could be a combination of fixed and/or removable storage devices, such as fixed disc drives, floppy disc drives, tape drives, removable memory cards, or optical storage. The memory 112 and storage 114 could be part of one virtual address space spanning multiple primary and secondary storage devices.

[0034] The memory 112 is preferably a random access memory (RAM) sufficiently large to hold the necessary programming and data structures of the invention. While the memory 112 is shown as a single entity, it should be understood that the memory 112 may in fact comprise a plurality of modules, and that the memory 112 may exist at multiple levels, from high speed registers and caches to lower speed but larger DRAM chips.

[0035] Illustratively, the memory 112 contains an operating system 124. Examples of suitable operating systems, which may be used to advantage, include Linux and Microsoft's Windows®, as well as any operating systems designed for handheld devices, such as Palm OS®, Windows® CE, and the like. More generally, any operating system supporting the functions disclosed herein may be used.

[0036] The memory 112 is also shown containing a query building interface 122, such as a browser program, that, when executed on CPU 110, provides support for building queries based on the data repository abstraction component 148. In one embodiment, the query interface 122 includes a web-based Graphical User Interface (GUI), which allows the user to display Hyper Text Markup Language (HTML) information. Functionality of the query interface 122 may be enhanced through the availability of one or more plug-in components (shown as plug-ins 129).

[0037] The plug-ins 162, XML configuration file 160, and result set data objects 165 are illustratively implemented on the server computer 104, while the interface 122 is implemented on the client computer 102. All of these system components – plug-ins 162, XML configuration files 160, result set data objects 165, and interfaces 122 – may be implemented or executed or both on any internal or external clients 102 of a networked system and be available to users (including applications) on any of the clients 102.

[0038] The server 104 may be physically arranged in a manner similar to the client computer 102. Accordingly, the server 104 is shown generally comprising a CPU 130, a memory 132, and a storage device 134, coupled to one another by a bus 136. Memory 132 may be a random access memory sufficiently large to hold the necessary programming and data structures that are located on the server 104.

[0039] The server 104 is generally under the control of an operating system 138 shown residing in memory 132. Examples of the operating system 138 include IBM OS/400®, UNIX, Microsoft Windows®, and the like. More generally, any operating system capable of supporting the functions described herein may be used. As illustrated, the server 104 may be configured with an abstract query interface 146 for issuing abstract queries (e.g., received from the client application 120) against one or more of the databases 156.

[0040] In one embodiment, elements of a query are specified by a user through the query building interface 122 which may be implemented as a browser program presenting a set of GUI screens for building queries. The content of the GUI screens may be generated by application(s) 140. In a particular embodiment, the GUI content is hypertext markup language (HTML) content which may be rendered on the client computer systems 102 with the query building interface 122. Accordingly, the memory 132 may include a Hypertext Transfer Protocol (http) server process 138 (e.g., a web server) adapted to service requests from the client computer 102. For example, the server process 152 may respond to requests to access the database(s) 156, which illustratively resides on the server 104. Incoming client requests for data from a database 156 invoke an application 140 which, when executed by the processor 130, perform operations necessary to access the database(s) 156. In one embodiment, the application 140 comprises a plurality of servlets configured to build GUI elements, which are then rendered by the query interface 122.

[0041] As previously described, the application 140 may also present the user with one or more plug-ins 162, available via the query interface 122 (or some other interface). These plug-ins may include analysis plug-ins that can be used to process data as desired by the user. Some plug-ins 162 may also be multi-analysis plug-ins, or plug-ins that are used to call other plug-ins. For example, if the user needs to call four separate plug-ins, rather than calling each of the four plug-ins 162 individually, the user may choose to call a multi-analysis plug-in that, in turn, calls each of the four plug-ins. The user benefits from calling the multi-analysis plug-in because only one plug-in would need to be invoked, rather than four. Further, if the user calls each of the plug-ins 162 individually, an added step of data transformation or mapping would be needed. The multi-analysis plug-in, however, does not require data transformation or data mapping because result sets produced by plug-ins are packaged as result set data objects 165. Each plug-in 162 will be able to accept the result set data object 165 as an input parameter and when processing is complete, provide a result set data object 165 as output. As used herein, the term plug-in may also refer to multi-analysis plug-ins.

[0042] If multiple plug-ins (including multi-analysis plug-ins) need to be called, the sequence in which the plug-ins are invoked is based on the contents of the XML configuration file 160. Aside from containing metadata associated with the plug-ins, the XML configuration file 160 also contains instructions relating to the execution of the plug-ins, including directives outlining the sequence in which plug-ins should be executed. The use of either an explicit sequence or a derived sequence may be specified. Explicit sequences are chosen by users and registered in the XML configuration file 160 along with the plug-in 162 metadata when the plug-in 162 is added to the system. Derived sequences are determined at runtime based on various factors including available result set data objects 165 for use as input parameters, and other system attributes.

AN EXEMPLARY RUNTIME ENVIRONMENT

[0043] Before describing the process of invoking and integrating multiple plug-ins 162 in detail, however, operation of the various illustrated components of the system will be described with reference to FIGs 2A and 2B. FIG 2A illustrates a relational view of a client application 120, a query 220 and the query execution runtime 150, according to one embodiment of the invention. As shown, the application 120 may be used to build a query 220 as designed by the user via the query building interface 122. Once built, the query is submitted to the query execution runtime 150.

[0044] For some embodiments, the query 220 may be an abstract query including a set of one or more query conditions and a specified results set, each based on logical fields defined in the DRA component 148 (shown in FIG. 1). As previously described, abstract queries may be executed by the query execution component 150. In the exemplary abstract data model, the logical fields are defined independently of the underlying data representation being used in the DBMS 154, thereby allowing queries to be formed that are loosely coupled to the underlying data representation 214. The query execution component 150 is generally configured to transform abstract queries into concrete queries compatible with the physical data representation (e.g., an XML representation 214_1, an SQL representation 214_2, or any other type of representation 214_3), by mapping the logical fields of the abstract queries to the corresponding physical fields of the physical data representation 214. The mapping of abstract queries to concrete queries, by the query execution component 150, is described in detail in the commonly owned, co-pending U.S. patent application 10/083,075, entitled “Application Portability And Extensibility Through Database Schema And Query Abstraction,” filed February 26, 2002.

[0045] FIG 2B illustrates the relational view of the application 120, after the query 220 described in FIG 2A is executed and the query result set 222 is returned to the application 120. The application 120 may then invoke specific plug-ins 162 to perform operations as desired by users. If invocation of multiple plug-ins is required, the application 120 may invoke the appropriate multi-analysis plug-in 161, rather than invoking each required plug-in 162 individually. Once invoked, the multi-analysis plug-in 161 will manage the execution of all required plug-ins 162.

[0046] The application 120 may pass the multi-analysis plug-in 161 the newly acquired result set 222 along with other required input parameters. Each analysis plug-in 162 invoked by the multi-analysis plug-in 161, may utilize a generic interface or signature as described in commonly owned co-pending application, entitled “Dynamic Functional Module Availability,” filed herewith (Attorney Docket No. ROC920030277US1). Further, each plug-in can accept a result set data object 165 as an input parameter and produce a result set data object 165 as an output parameter. For example, a query relating to all the micro-array data for a given experiment, is built and submitted to the query execution runtime 150, via the query building interface 122. Further, the user desires that two subsequent operations be performed on the result set 222 returned by the query execution runtime 150. First, the micro-array contained in the result set 222 needs to be normalized using a normalization plug-in, P1. Once the array is normalized, the next operation will call another plug-in, P2, to rank the normalized genes. After processing is completed, P1 will produce a result set data object 165 for use as an input parameter for P2. P2 will also produce a result set data object 165 after its processing is completed.

INVOKING AND INTEGRATING MULTIPLE PLUG-INS

[0047] FIGs 3A and 3B are flow diagrams of exemplary operations 300 for the invocation of a series of plug-ins in an explicitly defined sequence (FIG 3A) or derived sequence (FIG 3B). The operations 300 may be described with reference to FIGs 2A and 2B and may be performed, for example, to further process a query's 220 result set 222. For some embodiments, the operations 300 may be performed to build a query or analyze query results or both and dynamically invoke the necessary plug-ins 162, in order to complete all processing specified by the user.

[0048] FIG 3A focuses on operations 300 for an explicitly defined sequence, and begins, at step 301, when a query is built and initiated by a user. After the query is built, at step 302, the user submits the query and receives the corresponding results. During step 303, the application 120 compiles the results set 222 data and results metadata in a results set data object 165 and places the object in a result set collection 309_A.

[0049] As mentioned before, the result set data object also contains detailed results information, or metadata, describing the query results 222, such as column names, datatypes of columns and number of records returned, may also be stored in the metadata. Further, results metadata may also include details of the content returned in the result set 222. In other words, the results metadata may indicate the specific data values returned in the result set.

[0050] At step 304, the application 120 references the XML configuration file 160 and determines which plug-ins 162 need to be invoked. If multiple plug-ins are needed, the appropriate multi-analysis plug-ins 161 are called. Further, based on registered plug-in definitions in the XML configuration file 160, it is determined if an explicit or derived sequence for plug-in 162 execution is specified. The remainder of operations in FIG. 3A relate to an explicit sequence. First, the proper sequence is extracted from the XML configuration file 160 and passed to the multi-analysis plug-in 161 along with the result set data object 165. Illustratively, Table 1 below, shows an excerpt from a sample XML configuration file which specifies an explicit sequence for plug-in 162 execution. The sample XML configuration file contains an explicit sequence in which plug-ins (PLUGIN#1 and PLUGIN#2) may be invoked by a multi-analysis plug-in.

TABLE I – EXPLICIT SEQUENCE EXAMPLE

```
<Plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DQAPluginV1R3.xsd">

  <Extension className="com.ibm.dqa.plugin.analysis.PLUGIN#1" hidden="Yes"
name=" PLUGIN#1" point="com.ibm.dqa.plugin.analysis" deterministic="Yes">
          <Parms>
                  <Field name="NoOpParm" hidden="Yes">
                          <Type baseType="char"></Type>
                          <Description>Hidden extra info</Description>
                          <Value val="Example Data" />
                  </Field>
          </Parms>
          <PluginDesc>Test plugin - this has only hidden
parameters.</PluginDesc>
  </Extension>

  <Extension className="com.ibm.dqa.plugin.analysis.PLUGIN#2" hidden="No"
name="PLUGIN#2" point="com.ibm.dqa.plugin.analysis" deterministic="No">
          <Parms>
                  <Field name="FakeParm1">
                          <Type baseType="char"></Type>
                          <Description></Description>
                          <Value val="Example Default" />
                  </Field>
                  <Field name="FakeParm2">
                        <Type baseType="int"></Type>
                          <Description></Description>
                          <Value val="1" />
                  </Field>
          </Parms>
          <PluginDesc>This is an example plugin</PluginDesc>
  </Extension>

  <Extension name="MULTI-ANALYSIS PLUG-IN"
point="com.ibm.dqa.plugin.analysis"
         deterministic="No" hidden="No">
         <Steps>
                <Step name="step1" extension="PLUGIN#1" input="parent"/>
                <Step name="step2" extension="PLUGIN#2" input="step1">
```

```
                        <Parms>
                                  <Field name="ExampleParm1">
                                          <Type baseType="int"></Type>
                                          <Description></Description>
                                          <Value val="3" />
                                  </Field>
                                  <Field name="FakeParm2">
                                          <Type baseType="char"></Type>
                                          <Description></Description>
                                          <Value val="123456" />
                                  </Field>
                        </Parms>
                </Step>
                <Step name="step3" extension="PLUGIN#2" input="step2"/>
                <Output name="step3"/>
        </Steps>
          <PluginDesc>Runs several plugins in step</PluginDesc>
  </Extension>

</DQAPlugin>
```

[0051] At step 305, a loop of operations (306_A – 308) to be performed for each plug-in 162 is entered. At step 306A, the result set data object 165 is obtained from the result set collection 309A. Because the sequence of plug-in 162 invocation is known, it is not necessary to review the results metadata available in the results set data objects 165 to determine if requirements for input parameters are met. Instead, plug-ins 162 are simply run serially, as shown in step 307, in the order specified by the sequence. At step 308, after the current plug-in has completed processing, the result set data object 165 provided as an output parameter is place back into the result set collection 309_A. Once the loop of operations have been performed for each plug-in, the final results are returned to the application 120 and may be presented to the user via the query interface 122.

[0052] FIG 3B illustrates a flow diagram for the invocation of multiple plug-ins 126 in a derived sequence. The process of building a query, issuing the query, and placing the result set 222 (returned by the query execution runtime 150) in the result set collection 309_B are described by steps 301 - 303. These steps may be generally identical to the corresponding steps 301 - 303 in the explicit sequence based process described above with reference to FIG. 3A.

[0053] At step 304, as with the explicit process described above, the application refers to the XML configuration file 160 to generate a list of plug-ins 162 required to run. In this instance, however, the XML file specifies that the required plug-ins 162 be invoked in a derived sequence. Accordingly, the appropriate multi-analysis plug-ins 161 are called so that they, in turn, can invoke all required plug-ins 162 in the proper order. In addition, because use of a derived sequence is specified, the multi-analysis plug-in 161 will also need to utilize information regarding plug-in input and output parameters, included in the metadata associated with plug-ins 162, contained in the XML configuration file 160.

[0054] At step 305, a loop of operations (306B – 308) to be performed for each plug-in 162 is entered. At step 306B, the multi-analysis plug-in 161 chooses the next plug-in 162 to be invoked from a list of plug-ins 162 for which the available result set data objects 165, contained in the result set collection 309B, satisfy all input parameter requirements. The chosen plug-in 162 is then executed at step 307. At step 308, after processing is complete, the result set data object 165 produced by the plug-in 162 is made available for use by other plug-ins 162 by being placed in the result set collection 309B.

[0055] For example, the XML configuration file 160 may indicate that plug-ins 162 A1, A2, and A3, returning result sets RS1, RS2, and RS3 respectively, are required to run in a derived sequence. In addition, the XML configuration file 160 may indicate that the result set data objects 165 required as input for A1 include field F1. The field F3 is included in the output produced by A1. Similarly, field F2 is required by A2 and fields F4 and F5 are included in A2's output. Plug-in A3 requires fields F3 and F4 and provides output with fields F6 and F7. Further, the result set collection 309_B already contains a result set data object RS0 containing fields F1 and F2.

[0056] Accordingly, a multi-analysis plug-in 161 invokes plug-ins A1, A2, and A3. As described before, the multi-analysis plug-in 161 will manage the execution of all three plug-ins 162. Based on the information provided, it can be determined that the requirements of both A1 and A2 are satisfied by RS0, but A3 cannot be run because its input parameter requirements are not met. At this point, either A1 or A2 can be invoked. In addition, both plug-ins can be executed in parallel. After processing completes for A1 and A2, result sets RS1, containing field F3, and RS2, with fields F4 and F5, are produced and available for use by A3. All requirements of A3's input parameters are now satisfied, therefore, A3 is ready to be called.

CONCLUSION

[0057] By providing a method for efficiently invoking and integrating multiple functional modules, without requiring data transformation and data mapping, the user, by invoking just one functional module, may be able to invoke all selected functional modules. Accordingly, the user's experience with the application may be greatly enhanced.

[0058] While the foregoing is directed to embodiments of the present invention, other and further embodiments of the invention may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.